



EV546076579

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

A System and Related Methods
for
Reducing Source Filter Invocation in a Development
Project

Inventor(s):
Daniel J. Miller
Eric H. Rudolph

ATTORNEY'S DOCKET NO. MS1-631USC2



RELATED APPLICATION

This application is a divisional application of and claims priority to U.S. Patent Application Serial No. 09/731,490, the disclosure of which is incorporated by reference herein.

TECHNICAL FIELD

This invention generally relates to processing media content and, more particularly, to an interface and related methods for reducing source filter invocation during execution of a development project.

BACKGROUND

Recent advances in computing power and related technology have fostered the development of a new generation of powerful software applications. Gaming applications, communications applications, and multimedia applications have particularly benefited from increased processing power and clocking speeds. Indeed, once the province of dedicated, specialty workstations, many personal computing systems now have the capacity to receive, process and render multimedia objects (e.g., audio and video content). While the ability to display (receive, process and render) multimedia content has been around for a while, the ability for a standard computing system to support true multimedia editing applications is relatively new.

In an effort to satisfy this need, Microsoft Corporation introduced an innovative development system supporting advanced user-defined multimedia editing functions. An example of this architecture is presented in US Patent No.

1 5,913, 038 issued to Griffiths and commonly owned by the assignee of the present
2 invention, the disclosure of which is expressly incorporated herein by reference.

3 In the '038 patent, Griffiths introduced the an application program interface
4 which, when exposed to higher-level development applications, enable a user to
5 graphically construct a multimedia processing project by piecing together a
6 collection of "filters" exposed by the interface. The interface described therein is
7 referred to as a filter graph manager. The filter graph manager controls the data
8 structure of the filter graph and the way data moves through the filter graph. The
9 filter graph manager provides a set of software objects for communication
10 between a filter graph and its application. According to one implementation,
11 individual filters of a filter graph architecture are implemented as component
12 object model (COM) objects of an operating system executing on a computer,
13 each implementing one or more interfaces which contain a predefined set of
14 functions, called methods. Methods are called by an application program or other
15 component objects in order to communicate with the object exposing the interface.
16 The application program can also call methods or interfaces exposed by the filter
17 graph manager object.

18 Filter graphs work with data representing a variety of media (or non-media)
19 data types, each type characterized by a data stream that is processed by the filter
20 components comprising the filter graph. A filter positioned closer to the source of
21 the data is referred to as an upstream filter, while those further down the
22 processing chain is referred to as a downstream filter. For each data stream that
23 the filter handles it exposes at least one virtual pin (i.e., distinguished from a
24 physical pin such as one might find on an integrated circuit). A virtual pin can be
25 implemented as a COM object that represents a point of connection for a

1 unidirectional data stream on a filter. Input pins represent inputs and accept data
2 into the filter, while output pins represent outputs and provide data to other filters.
3 Each of the filters include at least one memory buffer, wherein communication of
4 the media stream between filters is accomplished by a series of “copy” operations
5 from one filter to another.

6 As introduced in Griffiths, a filter graph has three different types of filters:
7 source filters, transform filters, and rendering filters. A source filter is used to load
8 data from some source; a transform filter processes and passes data; and a
9 rendering filter renders data to a hardware device or other locations (e.g., saved to
10 a file, etc.). An example of a filter graph for a simplistic media rendering process
11 is presented with reference to Fig. 1.

12 Fig. 1 graphically illustrates an example filter graph for rendering media
13 content. As shown, the filter graph 100 is comprised of a plurality of filters 102-
14 114, which read, process (e.g., transform) and render media content from a
15 selected source file. As shown, the filter graph includes each of the types of filters
16 described above, interconnected in a linear fashion.

17 Products utilizing the filter graph have been well received in the market as
18 it has opened the door to multimedia editing using otherwise standard computing
19 systems. It is to be appreciated, however, that the construction and
20 implementation of the filter graphs are computationally intensive and expensive in
21 terms of memory usage. Even the most simple of filter graphs requires and
22 abundance of memory to facilitate the copy operations required to move data
23 between filters. Thus, complex filter graphs can become unwieldy, due in part to
24 the linear nature of conventional development system architecture. Moreover, it is
25

1 to be appreciated that the filter graphs themselves consume memory resources,
2 thereby compounding the issue introduced above.

3 Thus, what is required is a filter graph architecture which reduces the
4 computational and memory resources required to support even the most complex
5 of multimedia projects. Just such a solution is disclosed below.

6 **SUMMARY**

7
8 This invention concerns a system and related interfaces supporting the
9 processing of media content. In accordance with one aspect of the present
10 embodiment, a method of representing a development project is presented
11 comprising identifying a plurality of sources comprising the development project,
12 determining whether any of the sources are required simultaneously and, if not,
13 dynamically generating a filter graph representation of the development project
14 utilizing a segment filter to couple a source to multiple processing threads. In this
15 way, the source and source filter, typically coupled to each of the multiple
16 processing threads, is shared among multiple processing threads, reducing the
17 computational and memory requirements necessary to support execution of the
18 development project.

19 **BRIEF DESCRIPTION OF THE DRAWINGS**

20
21 The same reference numbers are used throughout the figures to reference
22 like components and features.

23 Fig. 1 is a graphical representation of a conventional filter graph
24 representing a user-defined development project.
25

1 Fig. 2 is a block diagram of a computing system incorporating the teachings
2 of the described embodiment.

3 Fig. 3 is a block diagram of an example software architecture incorporating
4 the teachings of the described embodiment.

5 Fig. 4 is a graphical illustration of an example software-enabled matrix
6 switch, according to an exemplary embodiment.

7 Fig. 5 is a graphical representation of a data structure comprising a
8 programming grid to selectively couple one or more of a scalable plurality of input
9 pins to a scalable plurality of output pins of the matrix switch filter, in accordance
10 with one aspect of the described embodiment.

11 Fig. 6 is a graphical illustration denoting shared buffer memory between
12 filters, according to one aspect of the described embodiment.

13 Fig. 7 is a flow chart of an example method for generating a filter graph, in
14 accordance with one aspect of the described embodiment.

15 Fig. 8 is a flow chart of an example method for negotiating buffer
16 requirements between at least two adjacent filters, according to one aspect of the
17 described embodiment.

18 Fig. 9 graphically illustrates an overview of a process that takes a user-
19 defined editing project and composites a data structure that can be used to program
20 the matrix switch.

21 Fig. 10 graphically illustrates the project of Fig. 9 in greater detail.

22 Fig. 11 shows an exemplary matrix switch dynamically generated in
23 support of the project developed in Figs. 9 and 10, according to one described
24 embodiment.
25

1 Fig. 12 illustrates a graphic representation of an exemplary data structure
2 that represents the project of Fig. 10, according to one described embodiment.

3 Figs. 13-18 graphically illustrate various states of a matrix switch
4 programming grid at select points in processing the project of Figs. 9 and 10
5 through the matrix switch, in accordance with one described embodiment.

6 Fig. 19 is a flow chart of an example method for processing media content,
7 in accordance with one described embodiment.

8 Fig. 20 illustrates an example project with a transition and an effect, in
9 accordance with one described embodiment.

10 Fig. 21 shows an exemplary data structure in the form of a hierarchical tree
11 that represents the project of Fig. 20.

12 Figs. 22 and 23 graphically illustrate an example matrix switch
13 programming grid associated with the project of Fig. 20 at select points in time,
14 according to one described embodiment.

15 Fig. 24 shows an example matrix switch dynamically generated and
16 configured as the grid of Figs. 22 and 23 was being processed, in accordance with
17 one described embodiment.

18 Fig. 25 shows an exemplary project in accordance with one described
19 embodiment.

20 Fig. 26 graphically illustrates an example audio editing project, according
21 to one described embodiment.

22 Fig. 27 depicts an example matrix switch programming grid associated with
23 the project of Fig. 26.
24
25

1 Fig. 28 shows an example matrix switch dynamically generated and
2 configured in accordance with the programming grid of Fig. 27 to perform the
3 project of Fig. 26, according to one described embodiment.

4 Fig. 29 illustrates an exemplary media processing project incorporating
5 another media processing project as a composite, according to yet another
6 described embodiment.

7 Fig. 30 graphically illustrates an example data structure in the form of a
8 hierarchical tree structure that represents the project of Fig. 29.

9 Figs 31-36 graphically illustrate various matrix switch programming grid
10 states at select points in generating and configuring the matrix switch to
11 implement the media processing of Fig. 29.

12 Fig. 38 illustrates an example matrix switch suitable for use in the media
13 processing project of Fig. 29, according to one described embodiment.

14 Fig. 38a graphically illustrates an example data structure in the form of a
15 hierarchical tree structure that represents a project that is useful in understanding
16 composites in accordance with the described embodiments.

17 Fig. 39 is a flow diagram that describes steps in a method in accordance
18 with one described embodiment.

19 Fig. 40 illustrates an example method of generating a filter graph, in
20 accordance with one aspect of the present invention.

21 Fig. 41 graphically illustrates an example reuse list, according to one aspect
22 of the present invention.

23 Fig. 42 illustrates an example method for source combining in support of
24 the method introduced in Fig. 40, according to one embodiment of the present
25 invention.

1 Fig. 43 graphically illustrates a timeline representation of source combining
2 introduced in Fig. 42.

3 Fig. 44 illustrates a block diagram of an example render engine utilizing a
4 segment filter in a filter graph, in accordance with one aspect of the present
5 invention.

6 Fig. 45 illustrates a flow chart of an example method of generating a filter
7 graph to reuse source filters, in accordance with one aspect of the present
8 invention.

9 Fig. 46 illustrates a flow chart of an example method of executing a
10 development project utilizing a segment filter, in accordance with one aspect of
11 the present invention.

12 **DETAILED DESCRIPTION**

13 **Related Applications**

14 This application is related to the following commonly-filed U.S. Patent
15 Applications, all of which are commonly assigned to Microsoft Corp., the
16 disclosures of which are incorporated by reference herein:
17

- 18 • Application Serial No. 09/731,560, entitled “An Interface and
19 Related Methods for Reducing Source Accesses in a Development
20 System”, naming Daniel J. Miller and Eric H. Rudolph as inventors;
- 21 • Application Serial No. 09/732,084, entitled “A System and Related
22 Interfaces Supporting the Processing of Media Content”, naming
23 Daniel J. Miller and Eric H. Rudolph as inventors;
- 24
- 25

- 1 • Application Serial No. 09/732,452, entitled “A System and Related
2 Methods for Reducing Memory Requirements of a Media Processing
3 System”, naming Daniel J. Miller and Eric H. Rudolph as inventors;
- 4 • Application Serial No. 09/731,529, entitled “A System and Related
5 Methods for Reducing the Instances of Source Files in a Filter
6 Graph”, naming Daniel J. Miller and Eric H. Rudolph as inventors;
- 7 • Application Serial No. 09/732,087, entitled “An Interface and
8 Related Methods for Dynamically Generating a Filter Graph in a
9 Development System”, naming Daniel J. Miller and Eric H. Rudolph
10 as inventors;
- 11 • Application Serial No. 09/732,090, entitled “A System and Related
12 Methods for Processing Audio Content in a Filter Graph”, naming
13 Daniel J. Miller and Eric H. Rudolph as inventors, and now U.S.
14 Patent No. 6,611,215;
- 15 • Application Serial No. 09/732,085, entitled “A System and Methods
16 for Generating an Managing Filter Strings in a Filter Graph”, naming
17 Daniel J. Miller and Eric H. Rudolph as inventors;
- 18 • Application Serial No. 09/731,491, entitled “Methods and Systems
19 for Processing Media Content”, naming Daniel J. Miller and Eric H.
20 Rudolph as inventors;
- 21 • Application Serial No. 09/731,563, entitled “Systems for Managing
22 Multiple Inputs and Methods and Systems for Processing Media
23 Content ”, naming Daniel J. Miller and Eric H. Rudolph as
24 inventors;
- 25 • Application Serial No. 09/731,892, entitled “Methods and Systems
for Implementing Dynamic Properties on Objects that Support Only
Static Properties”, naming Daniel J. Miller and David Maymudes as
inventors;

- 1 • Application Serial No. 09/732,089, entitled “Methods and Systems
2 for Efficiently Processing Compressed and Uncompressed Media
3 Content”, naming Daniel J. Miller and Eric H. Rudolph as inventors;
- 4 • Application Serial No. 09/731,581, entitled “Methods and Systems
5 for Effecting Video Transitions Represented By Bitmaps”, naming
6 Daniel J. Miller and David Maymudes as inventors;
- 7 • Application Serial No. 09/732,372, entitled “Methods and Systems
8 for Mixing Digital Audio Signals”, naming Eric H. Rudolph as
9 inventor; and
- 10 • Application Serial No. 09/732,086, entitled “Methods and Systems
11 for Processing Multi-media Editing Projects”, naming Eric H.
12 Rudolph as inventor.

13 Various described embodiments concern an application program interface
14 associated with a development system. According to one example
15 implementation, the interface is exposed to a media processing application to
16 enable a user to dynamically generate complex media processing tasks, e.g.,
17 editing projects. In the discussion herein, aspects of the invention are developed
18 within the general context of computer-executable instructions, such as program
19 modules, being executed by one or more conventional computers. Generally,
20 program modules include routines, programs, objects, components, data structures,
21 etc. that perform particular tasks or implement particular abstract data types.
22 Moreover, those skilled in the art will appreciate that the invention may be
23 practiced with other computer system configurations, including hand-held devices,
24 personal digital assistants, multiprocessor systems, microprocessor-based or
25 programmable consumer electronics, network PCs, minicomputers, mainframe
computers, and the like. In a distributed computer environment, program modules

1 may be located in both local and remote memory storage devices. It is noted,
2 however, that modification to the architecture and methods described herein may
3 well be made without deviating from spirit and scope of the present invention.
4 Moreover, although developed within the context of a media processing system
5 paradigm, those skilled in the art will appreciate, from the discussion to follow,
6 that the application program interface may well be applied to other development
7 system implementations. Thus, the media processing system described below is
8 but one illustrative implementation of a broader inventive concept.

9 10 **Example System Architecture**

11 **Fig. 2** illustrates an example of a suitable computing environment 200 on
12 which the system and related methods for processing media content may be
13 implemented.

14 It is to be appreciated that computing environment 200 is only one example
15 of a suitable computing environment and is not intended to suggest any limitation
16 as to the scope of use or functionality of the media processing system. Neither
17 should the computing environment 200 be interpreted as having any dependency
18 or requirement relating to any one or combination of components illustrated in the
19 exemplary computing environment 200.

20 The media processing system is operational with numerous other general
21 purpose or special purpose computing system environments or configurations.
22 Examples of well known computing systems, environments, and/or configurations
23 that may be suitable for use with the media processing system include, but are not
24 limited to, personal computers, server computers, thin clients, thick clients, hand-
25 held or laptop devices, multiprocessor systems, microprocessor-based systems, set

1 top boxes, programmable consumer electronics, network PCs, minicomputers,
2 mainframe computers, distributed computing environments that include any of the
3 above systems or devices, and the like.

4 In certain implementations, the system and related methods for processing
5 media content may well be described in the general context of computer-
6 executable instructions, such as program modules, being executed by a computer.
7 Generally, program modules include routines, programs, objects, components,
8 data structures, etc. that perform particular tasks or implement particular abstract
9 data types. The media processing system may also be practiced in distributed
10 computing environments where tasks are performed by remote processing devices
11 that are linked through a communications network. In a distributed computing
12 environment, program modules may be located in both local and remote computer
13 storage media including memory storage devices.

14 In accordance with the illustrated example embodiment of Fig. 2 computing
15 system 200 is shown comprising one or more processors or processing units 202, a
16 system memory 204, and a bus 206 that couples various system components
17 including the system memory 204 to the processor 202.

18 Bus 206 is intended to represent one or more of any of several types of bus
19 structures, including a memory bus or memory controller, a peripheral bus, an
20 accelerated graphics port, and a processor or local bus using any of a variety of
21 bus architectures. By way of example, and not limitation, such architectures
22 include Industry Standard Architecture (ISA) bus, Micro Channel Architecture
23 (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association
24 (VESA) local bus, and Peripheral Component Interconnects (PCI) bus also
25 known as Mezzanine bus.

1 Computer 200 typically includes a variety of computer readable media.
2 Such media may be any available media that is locally and/or remotely accessible
3 by computer 200, and it includes both volatile and non-volatile media, removable
4 and non-removable media.

5 In Fig. 2, the system memory 204 includes computer readable media in the
6 form of volatile, such as random access memory (RAM) 210, and/or non-volatile
7 memory, such as read only memory (ROM) 208. A basic input/output system
8 (BIOS) 212, containing the basic routines that help to transfer information
9 between elements within computer 200, such as during start-up, is stored in ROM
10 208. RAM 210 typically contains data and/or program modules that are
11 immediately accessible to and/or presently be operated on by processing unit(s)
12 202.

13 Computer 200 may further include other removable/non-removable,
14 volatile/non-volatile computer storage media. By way of example only, Fig. 2
15 illustrates a hard disk drive 228 for reading from and writing to a non-removable,
16 non-volatile magnetic media (not shown and typically called a "hard drive"), a
17 magnetic disk drive 230 for reading from and writing to a removable, non-volatile
18 magnetic disk 232 (e.g., a "floppy disk"), and an optical disk drive 234 for reading
19 from or writing to a removable, non-volatile optical disk 236 such as a CD-ROM,
20 DVD-ROM or other optical media. The hard disk drive 228, magnetic disk drive
21 230, and optical disk drive 234 are each connected to bus 206 by one or more
22 interfaces 226.

23 The drives and their associated computer-readable media provide
24 nonvolatile storage of computer readable instructions, data structures, program
25 modules, and other data for computer 200. Although the exemplary environment

1 described herein employs a hard disk 228, a removable magnetic disk 232 and a
2 removable optical disk 236, it should be appreciated by those skilled in the art that
3 other types of computer readable media which can store data that is accessible by a
4 computer, such as magnetic cassettes, flash memory cards, digital video disks,
5 random access memories (RAMs), read only memories (ROM), and the like, may
6 also be used in the exemplary operating environment.

7 A number of program modules may be stored on the hard disk 228,
8 magnetic disk 232, optical disk 236, ROM 208, or RAM 210, including, by way of
9 example, and not limitation, an operating system 214, one or more application
10 programs 216 (e.g., multimedia application program 224), other program modules
11 218, and program data 220. In accordance with the illustrated example
12 embodiment of Fig. 2, operating system 214 includes an application program
13 interface embodied as a render engine 222. As will be developed more fully
14 below, render engine 222 is exposed to higher-level applications (e.g., 216) to
15 automatically assemble filter graphs in support of user-defined development
16 projects, e.g., media processing projects. Unlike conventional media processing
17 systems, however, render engine 222 utilizes a scalable, dynamically
18 reconfigurable matrix switch to reduce filter graph complexity, thereby reducing
19 the computational and memory resources required to complete a development
20 project. Various aspects of the innovative media processing system represented by
21 a computer 200 implementing the innovative render engine 222 will be developed
22 further, below.

23 Continuing with Fig. 2, a user may enter commands and information into
24 computer 200 through input devices such as keyboard 238 and pointing device 240
25 (such as a "mouse"). Other input devices may include a audio/video input

1 device(s) 253, a microphone, joystick, game pad, satellite dish, serial port, scanner,
2 or the like (not shown). These and other input devices are connected to the
3 processing unit(s) 202 through input interface(s) 242 that is coupled to bus 206,
4 but may be connected by other interface and bus structures, such as a parallel port,
5 game port, or a universal serial bus (USB).

6 A monitor 256 or other type of display device is also connected to bus 206
7 via an interface, such as a video adapter 244. In addition to the monitor, personal
8 computers typically include other peripheral output devices (not shown), such as
9 speakers and printers, which may be connected through output peripheral interface
10 246.

11 Computer 200 may operate in a networked environment using logical
12 connections to one or more remote computers, such as a remote computer 250.
13 Remote computer 250 may include many or all of the elements and features
14 described herein relative to computer 200 including, for example, render engine
15 222 and one or more development applications 216 utilizing the resources of
16 render engine 222.

17 As shown in Fig. 2, computing system 200 is communicatively coupled to
18 remote devices (e.g., remote computer 250) through a local area network (LAN)
19 251 and a general wide area network (WAN) 252. Such networking environments
20 are commonplace in offices, enterprise-wide computer networks, intranets, and the
21 Internet.

22 When used in a LAN networking environment, the computer 200 is
23 connected to LAN 251 through a suitable network interface or adapter 248. When
24 used in a WAN networking environment, the computer 200 typically includes a
25 modem 254 or other means for establishing communications over the WAN 252.

1 The modem 254, which may be internal or external, may be connected to the
2 system bus 206 via the user input interface 242, or other appropriate mechanism.

3 In a networked environment, program modules depicted relative to the
4 personal computer 200, or portions thereof, may be stored in a remote memory
5 storage device. By way of example, and not limitation, Fig. 2 illustrates remote
6 application programs 216 as residing on a memory device of remote computer
7 250. It will be appreciated that the network connections shown and described are
8 exemplary and other means of establishing a communications link between the
9 computers may be used.

10 Turning next to **Fig. 3**, a block diagram of an example development system
11 architecture is presented, in accordance with one embodiment of the present
12 invention. In accordance with the illustrated example embodiment of Fig. 3,
13 development system 300 is shown comprising one or more application program(s)
14 216 coupled to render engine 222 via an appropriate communications interface
15 302. As used herein, application program(s) 216 are intended to represent any of a
16 wide variety of applications which may benefit from use of render engine 222
17 such as, for example a media processing application 224.

18 The communications interface 302 is intended to represent any of a number
19 of alternate interfaces used by operating systems to expose application program
20 interface(s) to applications. According to one example implementation, interface
21 302 is a component object model (COM) interface, as used by operating systems
22 offered by Microsoft Corporation. As introduced above, COM interface 302
23 provides a means by which the features of the render engine 222, to be described
24 more fully below, are exposed to an application program 216.

1 In accordance with the illustrated example implementation of Fig. 3, render
2 engine 222 is presented comprising source filter(s) 304A-N, transform filter(s)
3 306A-N and render filter 310, coupled together utilizing virtual pins to facilitate a
4 user-defined media processing project. According to one implementation, the
5 filters of system 300 are similar to the filters exposed in conventional media
6 processing systems. According to one implementation, however, filters are not
7 coupled via such interface pins. Rather, alternate implementations are envisioned
8 wherein individual filters (implemented as objects) make calls to other objects,
9 under the control of the render engine 222, for the desired input. Unlike
10 conventional systems, however, render engine 222 exposes a scalable, dynamically
11 reconfigurable matrix switch filter 308, automatically generated and dynamically
12 configured by render engine 222 to reduce the computational and memory
13 resource requirements often associated with development projects. As introduced
14 above, the pins (input and/or output) are application interface(s) designed to
15 communicatively couple other objects (e.g., filters).

16 In accordance with the example implementation of a media processing
17 system, an application communicates with an instance of render engine 222 when
18 the application 216 wants to process streaming media content. Render engine 222
19 selectively invokes and controls an instance of filter graph manager (not shown) to
20 automatically create a filter graph by invoking the appropriate filters (e.g., source,
21 transform and rendering). As introduced above, the communication of media
22 content between filters is achieved by either (1) coupling virtual output pins of one
23 filter to the virtual input pins of requesting filter; or (2) by scheduling object calls
24 between appropriate filters to communicate the requested information. As shown,
25 source filter 304 receives streaming data from the invoking application or an

1 external source (not shown). It is to be appreciated that the streaming data can be
2 obtained from a file on a disk, a network, a satellite feed, an Internet server, a
3 video cassette recorder, or other source of media content. As introduced above,
4 transform filter(s) 306 take the media content and processes it in some manner,
5 before passing it along to render filter 310. As used herein, transform filter(s) 306
6 are intended to represent a wide variety of processing methods or applications that
7 can be performed on media content. In this regard, transform filter(s) 306 may
8 well include a splitter, a decoder, a sizing filter, a transition filter, an effects filter,
9 and the like. The function of each of these filters is described more fully in the
10 Griffiths application, introduced above, and generally incorporated herein by
11 reference. The transition filter, as used herein, is utilized by render engine 222 to
12 transition the rendered output from a first source to a second source. The effect
13 filter is selectively invoked to introduce a particular effect (e.g., fade, wipe, audio
14 distortion, etc.) to a media stream.

15 In accordance with one aspect of the embodiment, to be described more
16 fully below, matrix switch filter 308 selectively passes media content from one or
17 more of a scalable plurality of input(s) to a scalable plurality of output(s).
18 Moreover, matrix switch 308 also supports implementation of a cascaded
19 architecture utilizing feedback paths, i.e., wherein transform filters 306B, 306C,
20 etc. coupled to the output of matrix switch 308 are dynamically coupled to one or
21 more of the scalable plurality of matrix switch input(s). An example of this
22 cascaded filter graph architecture is introduced in Fig. 3, and further explained in
23 example implementations, below.

24 Typically, media processed through source, transform and matrix switch
25 filters are ultimately passed to render filter 310, which provides the necessary

1 interface to a hardware device, or other location that accepts the renderer output
2 format, such as a memory or disk file, or a rendering device.

3 **Fig. 4** is a graphical illustration of an example software-enabled matrix
4 switch 308, according to one example embodiment of the present invention. As
5 shown, the matrix switch 308 is comprised of a scalable plurality of input(s) 402
6 and a scalable plurality of output(s) 404, wherein any one or more of the input(s)
7 402 may be iteratively coupled to any one or more of the output(s) 404, based on
8 the content of the matrix switch programming grid 406, automatically generated
9 by render engine 222. According to an alternate implementation introduced
10 above, switch matrix 308 is programmed by render engine 222 to dynamically
11 generate object calls to communicate media content between filters. In addition,
12 according to one implementation, matrix switch 308 includes a plurality of
13 input/output (I/O) buffers 408, as well as means for maintaining source, or media
14 time 410 and/or timeline, or project time 412. It is to be appreciated, however,
15 that in alternate implementations matrix switch 308 does not maintain both source
16 and project times, relying on an upstream filter to convert between these times. As
17 will be developed more fully below, matrix switch 308 dynamically couples one or
18 more of the scalable plurality of inputs 402 to one or more of the scalable plurality
19 of outputs 404 based, at least in part, on the media time 410 and/or the project time
20 412 and the content of matrix switch programming grid 406. In this regard, matrix
21 switch 308 may be characterized as time-aware, supporting such advanced editing
22 features as searching/seeking to a particular point (e.g., media time) in the media
23 content, facilitating an innovative buffering process utilizing I/O buffers 408 to
24 facilitate look-ahead processing of media content, and the like. Thus, it will be
25 appreciated given the discussion to follow that introduction of the matrix switch

1 308 provides a user with an editing flexibility that was heretofore unavailable in a
2 personal computer-based media processing system.

3 As introduced above, the inputs 402 and outputs 404 of matrix switch 308
4 are interfaces which facilitate the time-sensitive routing of data (e.g., media
5 content) in accordance with a user-defined development project. Matrix switch
6 308 has a scalable plurality of inputs 402 and outputs 404, meaning that the
7 number of inputs 402 and outputs 404 are individually generated to satisfy a given
8 editing project. Insofar as each of the inputs/outputs (I/O) has an associated
9 transfer buffer (preferably shared with an adjacent filter) to communicate media
10 content, the scalability of the input/output serves to reduce the overall buffer
11 memory consumed by an editing project. According to one implementation,
12 output 1 is generally reserved as a primary output, e.g., coupled to a rendering
13 filter (not shown).

14 According to one implementation, for each input 402 and output 404,
15 matrix switch 308 attempts to be the allocator, or manager of the buffer associated
16 with the I/O(s) shared with adjacent filters. One reason is to ensure that all of the
17 buffers are of the same size and share common attributes so that a buffer
18 associated with any input 402 may be shared with any output 404, thereby
19 reducing the need to copy memory contents between individual buffers associated
20 with such inputs/outputs. If matrix switch 308 cannot be an allocator for a given
21 output (404), communication from an input (402) to that output is performed using
22 a conventional memory copy operation between the individual buffers associated
23 with the select input/output.

24 As introduced above, the matrix switch programming grid 406 is
25 dynamically generated by render engine 222 based, at least in part, on the user-

1 defined development project. As will be developed below, render engine 222
2 invokes an instance of filter graph manager to assemble a tree structure of an
3 editing project, noting dependencies between source, filters and time to
4 dynamically generate the programming grid 406. A data structure comprising an
5 example programming grid 406 is introduced with reference to Fig. 5, below.

6 Turning briefly to **Fig. 5**, a graphical representation of a data structure
7 comprising an example programming grid 406 is presented, in accordance with
8 one embodiment of the present invention. In accordance with the illustrated
9 example embodiment of Fig. 5, programming grid 406 is depicted as a two-
10 dimensional data structure comprising a column along the y-axis 502 of the grid
11 denoting input pins associated with a content chain (e.g., series of filters to process
12 media content) of the development project. The top row along the x-axis 504 of
13 the data structure denotes project time. With these grid “borders”, the body 506 of
14 the grid 406 is populated with output pin assignments, denoting which input pin is
15 coupled to which output pin during execution of the development project. In this
16 way, render engine 222 dynamically generates and facilitates matrix switch 308.
17 Those skilled in the art will appreciate, however, that data structures of greater or
18 lesser complexity may well be used in support of the programming grid 406
19 without deviating from the spirit and scope of the present invention.

20 Returning to Fig. 4, matrix switch 308 is also depicted with a plurality of
21 input/output buffers 408, shared among all of the input(s)/output(s) (402, 404) to
22 facilitate advanced processing features. That is, while not required to implement
23 the core features of matrix switch 308, I/O buffers 408 facilitate a number of
24 innovative performance enhancing features to improve the performance (or at least
25 the user’s perception of performance) of the processing system, thereby providing

1 an improved user experience. According to one implementation, I/O buffers 408
2 are separate from the buffers assigned to each individual input and output pin in
3 support of communication through the switch. According to one implementation,
4 I/O buffers 408 are primarily used to foster look-ahead processing of the project.
5 Assume, for example, that a large portion of the media processing project required
6 only 50% of the available processing power, while some smaller portion required
7 150% of the available processing power. Implementation of the shared I/O buffers
8 408 enable filter graph manager to execute tasks ahead of schedule and buffer this
9 content in the shared I/O buffers 408 until required. Thus, when execution of the
10 filter graph reaches a point where more than 100% of the available processing
11 power is required, the processing system can continue to supply content from the
12 I/O buffers 408, while the system completes execution of the CPU-intensive tasks.
13 If enough shared buffer space is provided, the user should never know that some
14 tasks were not performed in real-time. According to one implementation, shared
15 buffers 408 are dynamically split into two groups by render engine 222, a first
16 group supports the input(s) 402, while a second (often smaller) group is used in
17 support of a primary output (e.g., output pin 1) to facilitate a second, independent
18 output processing thread. The use of an independent output buffers the render
19 engine from processing delays that might occur in upstream and/or downstream
20 filters, as discussed above. It will be appreciated by those skilled in the art that
21 such that matrix switch 308 and the foregoing described architecture beneficially
22 suited to support media streaming applications.

23 As introduced above, the filter graph is time-aware in the sense that media
24 (source) time and project execution time are maintained. According to one
25 implementation, matrix switch 308 maintains at least the project clock, while an

1 upstream filter maintains the source time, converting between source and project
2 time for all downstream filters (i.e., including the matrix switch 308). According
3 to one implementation, the frame rate converter filter of a filter graph is
4 responsible for converting source time to project time, and vice versa, i.e.,
5 supporting random seeks, etc. Alternatively, matrix switch 308 utilizes an
6 integrated set of clock(s) to independently maintain project and media times.

7 Having introduced the architectural and operational elements of matrix
8 switch filter 308, **Fig. 6** graphically illustrates an example filter graph
9 implementation incorporating the innovative matrix switch 308. In accordance
10 with the illustrated example embodiment, filter graph 600 is generated by render
11 engine 222 in response to a user defined development project. Unlike the lengthy
12 linear filter graphs typical of convention development systems however, filter
13 graph 600 is shown incorporating a matrix switch filter 308 to recursively route
14 the pre-processed content (e.g., through filters 602, 606, 610, 614 and 618,
15 described more fully below) through a user-defined number of transform filters
16 including, for example, transition filter(s) 620 and effects filter(s) 622. Moreover,
17 as will be developed more fully below, the scalable nature of matrix switch filter
18 308 facilitates such iterative processing for any number of content threads, tracks
19 or compositions.

20 According to one implementation, a matrix switch filter 308 can only
21 process one type of media content, of the same size and at the same frame-rate
22 (video) or modulation type/schema (audio). Thus, Fig. 6 is depicted comprising
23 pre-processing filters with a parser filter 606 to separate, independent content
24 type(s) (e.g., audio content and video content), wherein one of the media types
25 would be processed along a different path including a separate instance of matrix

1 switch 308. Thus, in accordance with the illustrated example embodiment of a
2 media processing system, processing multimedia content including audio and
3 video would utilize two (2) matrix switch filters 308, one dedicated to audio
4 processing (not shown) and one dedicated to video processing. That is not to say,
5 however, that multiple switch filters 308 could not be used (e.g., two each for
6 audio and video) for each content type in alternate implementations. Similarly, it
7 is anticipated that in alternate implementations a matrix switch 308 that accepts
8 multiple media types could well be used without deviating from the spirit and
9 scope of the present invention.

10 In addition filter graph 600 includes a decoder filter 610 to decode the
11 media content. Resize filter 614 is employed when matrix switch 308 is to receive
12 content from multiple sources, ensuring that the size of the received content is the
13 same, regardless of the source. According to one implementation, resize filter 614
14 is selectively employed in video processing paths to adjust the media size of
15 content from one or more sources to a user-defined level. Alternatively, resizer
16 filter 614 adjusts the media size to the largest size provided by any one or more
17 media sources. That is, if, for example, render engine 222 identifies the largest
18 required media size (e.g., 1270x1040 video pixels per frame) and, for any content
19 source not providing content at this size, the content is modified (e.g., stretched,
20 packed, etc.) to fill this size requirement. The frame rate converter (FRC) and
21 pack filter 618, introduced above, ensures that video content from the multiple
22 sources is arriving at the same frame rate, e.g., ten (10) frames per second. As
23 introduced above, the FRC also maintains the distinction between source time and
24 project time.
25

1 In accordance with one aspect of the present invention, filter graph 600 is
2 depicted utilizing a single, negotiated buffer 604, 608, 612, 616, etc. between
3 adjacent filters. In this regard, render engine 222 reduces the buffer memory
4 requirements in support of a development project.

5 From the point of pre-processing (filters 602, 606, 610, 614, 618), rather
6 than continue a linear filter graph incorporating all of the transition 620 and effect
7 622 filter(s), render engine 222 utilizes a cascade architecture, recursively passing
8 media content through the matrix switch 308 to apply to the transform filter(s)
9 (e.g., 620, 622, etc.) to complete the execution of the development project. It will
10 be appreciated by those skilled in the art that the ability to recursively pass media
11 content through one or more effect and/or transition filters provided by the matrix
12 switch filter 308 greatly reduces the perceived complexity of otherwise large filter
13 graphs, while reducing memory and computational overhead.

14 Turning to **Fig. 7**, a flow chart of an example method for generating a filter
15 graph is presented, in accordance with one aspect of the present invention. The
16 method 700 begins with block 702 wherein render engine 222 receives an
17 indication to generate a filter graph representing a user-defined development
18 project (e.g., a media editing project). According to one example implementation,
19 the indication is received from an application 224 via COM interface(s) 302.

20 In block 704, render engine 222 facilitates generation of the editing project,
21 identifying the number and type of media sources selected by the user. In block
22 706, based at least in part on the number and/or type of media sources, filter graph
23 manager 222 exposes source, transform and rendering filter(s) to effect a user
24 defined media processing project, while beginning to establish a programming
25 grid 406 for the matrix switch filter 308.

1 In block 708, reflecting user editing instructions, render engine 222
2 completes the programming grid 406 for matrix switch 308, identifying which
3 inputs 402 are to be coupled to which outputs 404 at particular project times.

4 Based, at least in part, on the programming grid 406 render engine 222
5 generates a matrix switch filter 308 with an appropriate number of input 402 and
6 output 404 pins to effect the project, and assembles the filter graph, block 710.

7 In block 712, to reduce the buffer memory requirements for the processing
8 project, the render engine 222 instructs the filters populating the filter graph to
9 (re)negotiate buffer memory requirements between filters. That is, adjacent filters
10 attempt to negotiate a size and attribute standard so that a single buffer can be
11 utilized to couple each an output pin of one filter to an input pin of a downstream
12 filter. An example implementation of the buffer negotiation process of block 712
13 is presented in greater detail with reference to Fig. 8.

14 Turning briefly to Fig. 8, an example method of negotiating buffer
15 requirements between adjacent filters is presented, in accordance with one
16 example implementation of the present invention. Once the final connection is
17 established to matrix switch 308, matrix switch 308 identifies the maximum buffer
18 requirements for any filter coupled to any of its pins (input 402 and/or output 404),
19 block 802. According to one implementation, the maximum buffer requirements
20 are defined as the lowest common multiple of buffer alignment requirements, and
21 the maximum of all the pre-fix requirements of the filter buffers.

22 In block 804, matrix switch 308 selectively removes one or more existing
23 filter connections to adjacent filters. Matrix switch 308 then reconnects all of its
24 pins to adjacent filters using a common buffer size between each of the pins, block
25 806. In block 808, matrix switch 308 negotiates to be the allocator for all of its

1 pins (402, 404). If the matrix switch 308 cannot, for whatever reason, be the
2 allocator for any of its input pins 402 minimal loss to performance is encountered,
3 as the buffer associated with the input pin will still be compatible with any
4 downstream filter (i.e., coupled to an output pin) and, thus, the buffer can still be
5 passed to the downstream filter without requiring a memory copy operation. If,
6 however, matrix switch 308 cannot be an allocator for one of its output pins 404,
7 media content must then be transferred to at least the downstream filter associated
8 with that output pin using a memory copy operation, block 810.

9 In block 812, once the matrix switch 308 has re-established its connection
10 to adjacent filters, render engine 222 restores the connection in remaining filters
11 using negotiated buffer requirements emanating from the matrix switch filter 308
12 buffer negotiations. Once the connections throughout the filter graph have been
13 reconnected, the process continues with block 714 of Fig. 7.

14 In block 714 (Fig. 7), have re-established the connections between filters,
15 render engine 222 is ready to implement a user's instruction to execute the media
16 processing project.

17 18 **Example Operation and Implementation(s)**

19 The matrix switch described above is quite useful in that it allows multiple
20 inputs to be directed to multiple outputs at any one time. These input can compete
21 for a matrix switch output. The embodiments described below permit these
22 competing inputs to be organized so that the inputs smoothly flow through the
23 matrix switch to provide a desired output. And, while the inventive programming
24 techniques are described in connection with the matrix switch as such is employed
25 in the context of multi-media editing projects, it should be clearly understood that

1 application of the inventive programming techniques and structures should not be
2 so limited only to application in the field of multi-media editing projects or, for
3 that matter, multi-media applications or data streams. Accordingly, the principles
4 about to be discussed can be applied to other fields of endeavor in which multiple
5 inputs can be characterized as competing for a particular output during a common
6 time period.

7 In the multi-media example below, the primary output of the matrix switch
8 is a data stream that defines an editing project that has been created by a user.
9 Recall that this editing project can include multiple different sources that are
10 combined in any number of different ways, and the sources that make up a project
11 can comprise audio sources, video sources, or both. The organization of the inputs
12 and outputs of the matrix switch are made manageable, in the examples described
13 below, by a data structure that permits the matrix switch to be programmed.

14 Fig. 9 shows an overview of a process that takes a user-defined editing
15 project and renders from it a data structure that can be used to program the matrix
16 switch.

17 Specifically, a user-defined editing project is shown generally at 900.
18 Typically, when a user creates an editing project, they can select from a number of
19 different multimedia clips that they can then assemble into a unique presentation.
20 Each individual clip represents a *source* of digital data or a source stream (e.g.,
21 multimedia content). Projects can include one or more sources 902. In defining
22 their project, a user can operate on sources in different ways. For example, video
23 sources can have *transitions* 904 and *effects* 906 applied on them. A transition
24 object is a way to change between two or more sources. As discussed above, a
25 transition essentially receives as input, two or more streams, operates on them in

1 some way, and produces a single output stream. An exemplary transition can
2 comprise, for example, fading from one source to another. An effect object can
3 operate on a single source or on a composite of sources. An effect essentially
4 receives a single input stream, operates on it in some way, and produces a single
5 output stream. An exemplary effect can comprise a black-and-white effect in
6 which a video stream that is configured for presentation in color format is
7 rendered into a video stream that is configured for presentation in black and white
8 format. Unlike conventional effect filters, effect object 906 may well perform
9 multiple effect tasks. That is, in accordance with one implementation, an effect
10 object (e.g., 906) may actually perform multiple tasks on the received input
11 stream, wherein said tasks would require multiple effect filters in a conventional
12 filter graph system.

13 An exemplary user interface 908 is shown and represents what a user might
14 see when they produce a multimedia project with software executing on a
15 computer. In this example, the user has selected three sources A, B, and C, and
16 has assembled the sources into a project timeline. The project timeline defines
17 when the individual sources are to be rendered, as well as when any transitions
18 and/or effects are to occur.

19 In the discussion that follows, the notion of a *track* is introduced. A track
20 can contain one or more sources or source clips. If a track contains more than one
21 source clip, the source clips cannot overlap. If source clips are to overlap (e.g.
22 fading from one source to another, or having one source obscure another), then
23 multiple tracks are used. A track can thus logically represent a layer on which
24 sequential video is produced. User interface 908 illustrates a project that utilizes
25 three tracks, each of which contains a different source. In this particular project

1 source A will show for a period of time. At a defined time in the presentation,
2 source A is obscured by source B. At some later time, source B transitions to
3 source C.

4 In accordance with the described embodiment, the user-defined editing
5 project 900 is translated into a data structure 910 that represents the project. In the
6 illustrated and described example, this data structure 910 comprises a tree
7 structure. It is to be understood, however, that other data structures could be used.
8 The use of tree structures to represent editing projects is well-known and is not
9 described here in any additional detail. Once the data structure 910 is defined, it is
10 processed to provide a data structure 912 that is utilized to program the matrix
11 switch. In the illustrated and described embodiment, data structure 912 comprises
12 a grid from which the matrix switch can be programmed. It is to be understood
13 and appreciated that other data structures and techniques could, however, be used
14 to program the matrix switch without departing from the spirit and scope of the
15 claimed subject matter.

16 The processing that takes place to define data structures 910 and 912 can
17 take place using any suitable hardware, software, firmware, or combination
18 thereof. In the examples set forth below, the processing takes place utilizing
19 software in the form of a video editing software package that is executable on a
20 general purpose computer.

21 22 Example Project

23 For purposes of explanation, consider Fig. 10 which shows project 908
24 from Fig. 9 in a little additional detail. Here, a time line containing numbers 0-16
25 is provided adjacent the project to indicate when particular sources are to be seen

and when transitions and effects (when present) are to occur. In the examples in this document, the following convention exists with respect to projects, such as project 908. A priority exists for video portions of the project such that as one proceeds from top to bottom, the priority increases. Thus, in the Fig. 10 example, source A has the lowest priority followed by source B and source C. Thus, if there is an overlap between higher and lower priority sources, the higher priority source will prevail. For example, source B will obscure source A from between $t = 4-8$.

In this example, the following can be ascertained from the project 908 and time line: from time $t=0-4$ source A should be routed to the matrix switch's primary output; from $t=4-12$ source B should be routed to the matrix switch's primary output; from $t=12-14$ there should be a transition between source B and source C which should be routed to the matrix switch's primary output; and from $t=14-16$ source C should be routed to the matrix switch's primary output. Thus, relative to the matrix switch, each of the sources and the transition can be characterized by where it is to be routed at any given time. Consider, for example, the table just below:

Object	Routing for a given time
C	$t = 0-12$ (nowhere); $t = 12-14$ (transition); $t = 14-16$ (primary output)
B	$t = 0-4$ (nowhere); $t = 4-12$ (primary output); $t = 12-14$ (transition); $t = 14-16$ (nowhere)
A	$t = 0-4$ (primary output); $t = 4-16$ (nowhere)
Transition	$t = 0-12$ (nowhere); $t = 12-14$ (primary output); $t = 14-16$ (nowhere)

1 Fig. 11 shows an exemplary matrix switch 1100 that can be utilized in the
2 presentation of the user's project. Matrix switch 1100 comprises multiple inputs
3 and multiple outputs. Recall that a characteristic of the matrix switch 1100 is that
4 any of the inputs can be routed to any of the outputs at any given time. A
5 transition element 1102 is provided and represents the transition that is to occur
6 between sources B and C. Notice that the matrix switch includes four inputs
7 numbered 0-3 and three outputs numbered 0-2. Inputs 0-2 correspond respectively
8 to sources A-C, while input 3 corresponds to the output of the transition element
9 1102. Output 0 corresponds to the switch's primary output, while outputs 1 and 2
10 are routed to the transition element 1102.

11 The information that is contained in the table above is the information that
12 is utilized to program the matrix switch. The discussion presented below describes
13 but one implementation in which the information contained in the above table can
14 be derived from the user's project time line.

15 Recall that as a user edits or creates a project, software that comprises a part
16 of their editing software builds a data structure that represents the project. In the
17 Fig. 9 overview, this was data structure 910. In addition to building the data
18 structure that represents the editing project, the software also builds and configures
19 a matrix switch that is to be used to define the output stream that embodies the
20 project. Building and configuring the matrix switch can include building the
21 appropriate graphs (e.g., a collection of software objects, or filters) that are
22 associated with each of the sources and associating those graphs with the correct
23 inputs of the matrix switch. In addition, building and configuring the matrix
24 switch can also include obtaining and incorporating additional appropriate filters
25

1 with the matrix switch, e.g. filters for transitions, effects, and mixing (for audio
2 streams). This will become more apparent below.

3 Fig. 12 shows a graphic representation of an exemplary data structure 1200
4 that represents the project of Fig. 10. Here, the data structure comprises a
5 traditional hierarchical tree structure. Any suitable data structure can, however, be
6 utilized. The top node 1202 constitutes a *group* node. A *group* encapsulates a type
7 of media. For example, in the present example the media type comprises video.
8 Another media type is audio. The group node can have child nodes that are either
9 tracks or composites. In the present example, three track nodes 1204, 1206, and
10 1208 are shown. Recall that each track can have one or more sources. If a track
11 comprises more than one source, the sources cannot overlap. Here, all of the
12 sources (A, B, and C) overlap. Hence, three different tracks are utilized for the
13 sources. In terms of priority, the lowest priority source is placed into the tree
14 furthest from the left at 1204a. The other sources are similarly placed. Notice that
15 source C (1208a) has a transition 1210 associated with it. A transition object, in
16 this example, defines a two-input/one output operation. When applied to a track
17 or a composition (discussed below in more detail), the transition object will
18 operate between the track to which it has been applied, and any objects that are
19 beneath it in priority and at the same level in the tree. A “tree level” has a
20 common depth within the tree and belongs to the same parent. Accordingly, in
21 this example, the transition 1210 will operate on a source to the left of the track on
22 which source C resides, and beneath it in priority, i.e. source B. If the transition is
23 applied to any object that has nothing beneath it in the tree, it will transition from
24 blackness (and/or silence if audio is included).

1 Once a data structure representing the project has been built, in this case a
2 hierarchical tree structure, a rendering engine processes the data structure to
3 provide another data structure that is utilized to program the matrix switch. In the
4 Fig. 9 example, this additional data structure is represented at 912. It will be
5 appreciated and understood that the nodes of tree 1200 can include so-called meta
6 information such as a name, ID, and a time value that represents when that
7 particular node's object desires to be routed to the output, e.g. node 1204a would
8 include an identifier for the node associating it with source A, as well as a time
9 value that indicates that source A desires to be routed to the output from time $t = 0$ -
10 8. This meta information is utilized to build the data structure that is, in turn,
11 utilized to program the matrix switch.

12 In the example about to be described below, a specific data structure in the
13 form of a grid is utilized. In addition, certain specifics are described with respect
14 to how the grid is processed so that the matrix switch can be programmed. It is to
15 be understood that the specific described approach is for exemplary purposes only
16 and is not intended to limit application of the claims. Rather, the specific approach
17 constitutes but one way of implementing broader conceptual notions embodied by
18 the inventive subject matter.

19 Figs. 13-18 represent a process through which the inventive grid is built. In
20 the grid about to be described, the x axis represents time, and the y axis represents
21 layers in terms of priority that go from lowest (at the top of the grid) to highest (at
22 the bottom of the grid). Every row in the grid represents the video layer.
23 Additionally, entries made within the grid represent output pins of the matrix
24 switch. This will become apparent below.
25

1 The way that the grid is built in this example is that the rendering engine
2 does a traversal operation on the tree 1200. In this particular example, the
3 traversal operation is known as a “depth-first, left-to-right” traversal. This
4 operation will layerize the nodes so that the leftmost track or source has the lowest
5 priority and so on. Doing the above-mentioned traversal on tree 1200 (Fig. 12),
6 the first node encountered is node 1204 which is associated with source A. This is
7 the lowest priority track or source. A first row is defined for the grid and is
8 associated with source A. After the first grid row is defined, a grid entry is made
9 and represents the time period for which source A desires to be routed to the
10 matrix switch’s primary output.

11 Fig. 13 shows the state of a grid 1300 after this first processing step.
12 Notice that from time $t = 0-8$, a “0” has been placed in the grid. The “0”
13 represents the output pin of the matrix switch—in this case the primary output.
14 Next, the traversal encounters node 1206 (Fig. 12) which is associated with source
15 B. A second row is thus defined for the grid and is associated with source B.
16 After the second grid row is defined, a grid entry is made and represents the time
17 period for which source B desires to be routed to the matrix switch’s primary
18 output.

19 Fig. 14 shows the state of grid 1300 after this second processing step.
20 Notice that from time $t = 4-14$, a “0” has been placed in the grid. Notice at this
21 point that something interesting has occurred which will be resolved below. Each
22 of the layers has a common period of time (i.e. $t = 4-8$) for which it desires to be
23 routed to the matrix switch’s primary output. However, because of the nature of
24 the matrix switch, only one input can be routed to the primary output at a time.
25 Next, the traversal encounters node 1208 (Fig. 12) which is associated with source

1 C. In this particular processing example, a rule is defined that sources on tracks
2 are processed before transitions on the tracks are processed because transitions
3 operate on two objects that are beneath them. A third row is thus defined for the
4 grid and is associated with source C. After the third row is defined, a grid entry is
5 made and represents the time period for which source C desires to be routed to the
6 matrix switch's primary output.

7 Fig. 15 shows the state of grid 1300 after this third processing step. Notice
8 that from time $t = 12-16$, a "0" has been placed in the grid. Next, the traversal
9 encounters node 1210 (Fig. 12) which corresponds to the transition. Thus, a fourth
10 row is defined in the grid and is associated with the transition. After the fourth
11 row is defined, a grid entry is made and represents the time period for which the
12 transition desires to be routed to the matrix switch's primary output.

13 Fig. 16 shows the state of grid 1300 after this fourth processing step.
14 Notice that from time $t = 12-14$, a "0" has been placed in the grid for the transition
15 entry. The transition is a special grid entry. Recall that the transition is
16 programmed to operate on two inputs and provide a single output. Accordingly,
17 starting at the transition entry in the grid and working backward, each of the
18 entries corresponding to the same tree level are examined to ascertain whether
19 they contain entries that indicate that they want to be routed to the output during
20 the same time that the transition is to be routed to the output. If grid entries are
21 found that conflict with the transition's grid entry, the conflicting grid entry is
22 changed to a value that corresponds to an output pin that serves as an input to the
23 transition element 1102 (Fig. 11). This is essentially a redirection operation. In
24 the illustrated grid example, the transition first finds the level that corresponds to
25 source C. This level conflicts with the transition's grid entry for the time period t

1 = 12-14. Thus, for this time period, the grid entry for level C is changed to a
2 switch output that corresponds to an input for the transition element. In this
3 example, a "2" is placed in the grid to signify that for this given time period, this
4 input is routed to output pin 2. Similarly, continuing up the grid, the next level
5 that conflicts with the transition's grid entry is the level that corresponds to source
6 B. Thus, for the conflicting time period, the grid entry for level B is changed to a
7 switch output that corresponds to an input for the transition element. In this
8 example, a "1" is placed in the grid to signify that for this given time period, this
9 input is routed to output pin 1 of the matrix switch.

10 Fig. 17 shows the state of the grid at this point in the processing. Next, a
11 pruning function is implemented which removes any other lower priority entry
12 that is contending for the output with a higher priority entry. In the example, the
13 portion of A from $t=4-8$ gets removed because the higher priority B wants the
14 output for that time.

15 Fig. 18 shows the grid with a cross-hatched area that signifies that portion
16 of A's grid entry that has been removed.

17 At this point, the grid is in a state in which it can be used to program the
18 matrix switch. The left side entries -- A, B, C, and TRANS represent input pin
19 numbers 0, 1, 2, and 3 (as shown) respectively, on the matrix switch shown in Fig.
20 11. The output pin numbers of the matrix switch are designated at 0, 1, and 2 both
21 on the switch in Fig. 11 and within the grid in Fig. 18. As one proceeds through
22 the grid, starting with source A, the programming of the matrix switch can be
23 ascertained as follows: A is routed to output pin 0 of the matrix switch (the
24 primary output) from $t = 0-4$. From $t = 4-16$, A is not routed to any output pins.
25 From $t = 0-4$, B is not routed to any of the output pins of the matrix switch. From t

1 = 4-12, B is routed to the primary output pin 0 of the matrix switch. From $t = 12$ -
2 14, B is routed to output pin 1 of the matrix switch. Output pin 1 of the matrix
3 switch corresponds to one of the input pins for the transition element 1102 (Fig.
4 11). From $t = 14$ -16, B is not routed to any of the output pins of the matrix switch.
5 From $t = 0$ -12, C is not routed to any of the output pins of the matrix switch. From
6 $t = 12$ -14, C is routed to output pin 2 of the matrix switch. Output pin 2 of the
7 matrix switch corresponds to one of the input pins for the transition element 302
8 (Fig. 3). From $t = 12$ -14 the transition element (input pin 3) is routed to output pin
9 0. From $t = 14$ -16, C is routed to output pin 0 of the matrix switch.

10 As alluded to above, one of the innovative aspects of the matrix switch 308
11 is its ability to seek to any point in a source, without having to process the
12 intervening content serially through the filter. Rather, matrix switch 308 identifies
13 an appropriate transition point and dumps at least a subset of the intervening
14 content, and continues processing from the sought point in the content.

15 The ability of the matrix switch 308 to seek to any point in the media
16 content gives rise to certain performance enhancement heretofore unavailable in
17 computer implemented media processing systems. For example, generation of a
18 filter graph by render engine 222 may take into account certain performance
19 characteristics of the media processing system which will execute the user-defined
20 media processing project. In accordance with this example implementation,
21 render engine 222 may access and analyze the system registry of the operating
22 system, for example, to ascertain the performance characteristics of hardware
23 and/or software elements of the computing system implementing the media
24 processing system, and adjust the filter graph construction to improve the
25 perceived performance of the media processing system by the user. Nonetheless,

1 there will always be a chance that a particular instance of a filter graph will not be
2 able to process the media stream fast enough to provide the desired output at the
3 desired time, i.e., processing of the media stream bogs down leading to delays at
4 the rendering filter. In such a case, matrix switch 308 will recognize that it is not
5 receiving media content at the appropriate project time, and may skip certain
6 sections of the project in an effort to “catch-up” and continue the remainder of the
7 project in real time. According to one implementation, when matrix switch 308
8 detects such a lag in processing, it will analyze the degree of the lag and issue a
9 seek command to the source (through the source processing chain) to a future
10 point in the project, where processing continues without processing any further
11 content prior to the sought point.

12 Thus, for the editing project depicted in Fig. 10, the processing described
13 above first builds a data structure (i.e. data structure 1200 in Fig. 12) that
14 represents the project in hierarchical space, and then uses this data structure to
15 define or create another data structure that can be utilized to program the matrix
16 switch.

17 Fig. 19 is a flow diagram that describes steps in a method in accordance
18 with the described embodiment. The method can be implemented in any suitable
19 hardware, software, firmware, or combination thereof. In the illustrated and
20 described embodiment, the method is implemented in software.

21 Step 1900 provides a matrix switch. An exemplary matrix switch is
22 described above. Step 1902 defines a first data structure that represents the editing
23 project. Any suitable data structure can be used, as will be apparent to those of
24 skill in the art. In the illustrated and described embodiment, the data structure
25 comprises a hierarchical tree structure having nodes that can represent tracks

1 (having one or more sources), composites, transitions and effects. Step 1904
2 processes the first data structure to provide a second data structure that is
3 configured to program the matrix switch. Any suitable data structure can be
4 utilized to implement the second data structure. In the illustrated and described
5 embodiment, a grid structure is utilized. Exemplary processing techniques for
6 processing the first data structure to provide the second data structure are
7 described above. Step 1906 then uses the second data structure to program the
8 matrix switch.

10 Example Project with a Transition and an Effect

11 Consider project 2000 depicted in Fig. 20. In this project there are three
12 tracks, each of which contains a source, i.e. source A, B and C. This project
13 includes an effect applied on source B and a transition between sources B and C.
14 The times are indicated as shown.

15 As the user creates their project, a data structure representing the project is
16 built. Fig. 21 shows an exemplary data structure in the form of a hierarchical tree
17 2100 that represents project 2000. There, the data structure includes three tracks,
18 each of which contains one of the sources. The sources are arranged in the tree
19 structure in the order of their priority, starting with the lowest priority source on
20 the left and proceeding to the right. There is an effect (i.e. "Fx") that is attached to
21 or otherwise associated with source B. Additionally, there is a transition attached
22 to or otherwise associated with source C.

23 In building the grid for project 2000, the following rule is employed for
24 effects. An effect, in this example, is a one-input/one-output object that is applied
25 to one object—in this case source B. When the effect is inserted into the grid, it

1 looks for any one object beneath it in priority that has a desire to be routed to the
2 primary output of the matrix switch at the same time. When it finds a suitable
3 object, it redirects that object's output from the matrix switch's primary output to
4 an output associated with the effect.

5 As an example, consider Fig. 22 and the grid 2200. At this point in the
6 processing of tree 2100, the rendering engine has incorporated entries in the grid
7 corresponding to sources A, B and the effect. It has done so by traversing the tree
8 2100 in the above-described way. In this example, the effect has already looked
9 for an object beneath it in priority that is competing for the primary output of the
10 matrix switch. It found an entry for source B and then redirected B's grid entry to
11 a matrix switch output pin that corresponds to the effect—here output pin 1.

12 As the render engine 222 completes its traversal of tree 2100, it completes
13 the grid. Fig. 23 shows a completed grid 2200. Processing of the grid after that
14 which is indicated in Fig. 22 takes place substantially as described above with
15 respect to the first example. Summarizing, this processing though: after the effect
16 is entered into the grid and processed as described above, the traversal of tree 2100
17 next encounters the node associated with source C. Thus, a row is added in the
18 grid for source C and an entry is made to indicate that source C desires the output
19 from $t = 12-16$. Next, the tree traversal encounters the node associated with the
20 transition. Accordingly, a row is added to the grid for the transition and a grid
21 entry is made to indicate that the transition desires the output from $t = 12-14$.
22 Now, as described above, the grid is examined to find two entries, lower in
23 priority than the transition and located at the same tree level as the transition, that
24 compete for the primary output of the matrix switch. Here, those entries
25 correspond to the grid entries for the effect and source C that occur from $t = 12-14$.

1 These grid entries are thus redirected to output pins of the matrix switch 308 that
2 correspond to the transition—here pins 2 and 3 as indicated. Next, the grid is
3 pruned which, in this example, removes a portion of the grid entry corresponding
4 to source A for $t = 4-8$ because of a conflict with the higher-priority entry for
5 source B.

6 Fig. 24 shows the resultant matrix switch that has been built and configured
7 as the grid was being processed above. At this point, the grid can be used to
8 program the matrix switch. From the grid picture, it is very easy to see how the
9 matrix switch 308 is going to be programmed. Source A will be routed to the
10 matrix switch's primary output (pin 0) from $t = 0-4$; source B will be redirected to
11 output pin 1 (effect) from $t = 4-14$ and the effect on B will be routed to the output
12 pin 0 from $t = 4-12$. From $t = 12-14$, the effect and source C will be routed to
13 output pins corresponding to the transition (pins 2 and 3) and, accordingly, during
14 this time the transition (input pin 4) will be routed to the primary output (output
15 pin 0) of the matrix switch. From $t = 14-16$, source C will be routed to the primary
16 output of the matrix switch.

17 It will be appreciated that as the software, in this case the render engine
18 222, traverses the tree structure that represents a project, it also builds the
19 appropriate graphs and adds the appropriate filters and graphs to the matrix switch.
20 Thus, for example, as the render engine 222 encounters a tree node associated with
21 source A, in addition to adding an entry to the appropriate grid, the software builds
22 the appropriate graphs (i.e. collection of linked filters), and associates those filters
23 with an input of the matrix switch. Similarly, when the render engine 222
24 encounters an effect node in the tree, the software obtains an effect object or filter
25 and associates it with the appropriate output of the matrix switch. Thus, in the

1 above examples, traversal of the tree structure representing the project also enables
2 the software to construct the appropriate graphs and obtain the appropriate objects
3 and associate those items with the appropriate inputs/outputs of the matrix switch
4 308. Upon completion of the tree traversal and processing of the grid, an
5 appropriate matrix switch has been constructed, and the programming (i.e. timing)
6 of inputs to outputs for the matrix switch has been completed.

7 8 **Treatment of “blanks” in a Project**

9 There may be instances in a project when a user leaves a blank in the
10 project time line. During this blank period, no video or audio is scheduled for
11 play.

12 Fig. 25 shows a project that has such a blank incorporated therein. If there
13 is such a blank left in a project, the software is configured to obtain a “black”
14 source and associate the source with the matrix switch at the appropriate input pin.
15 The grid is then configured when it is built to route the black source to the output
16 at the appropriate times and fade from the black (and silent) source to the next
17 source at the appropriate times. The black source can also be used if there is a
18 transition placed on a source for which there is no additional source from which to
19 transition.

20 21 **Audio Mixing**

22 In the examples discussed above, sources comprising video streams were
23 discussed. In those examples, at any one time, only two video streams were
24 combined into one video stream. However, each project can, and usually does
25 contain an audio component. Alternately, a project can contain only an audio

1 component. The audio component can typically comprise a number of different
2 audio streams that are combined. The discussion below sets forth but one way of
3 processing and combining audio streams.

4 In the illustrated example, there is no limit on the number of audio streams
5 that can be combined at any one time.

6 Suppose, for example, there is an audio project that comprises 5 tracks, A-
7 E. Fig. 26 shows an exemplary project. The shaded portions of each track
8 represent the time during which the track is not playing. So, for example, at $t=0-4$,
9 tracks B, D, and E are mixed together and will play. From $t = 4-10$, tracks A-E are
10 mixed together and will play, and the like.

11 Fig. 27 shows the grid for this project at 2700. Since we are dealing with
12 this composition now, all of the effects and transitions including the audio mixing
13 are only allowed to affect things in this composition. Thus, there is the concept of
14 a boundary 2702 that prevents any actions or operations in this composition from
15 affecting any other grid entries. Note that there are other entries in the grid and
16 that the presently-illustrated entries represent only those portions of the project
17 that relate to the audio mixing function.

18 Grid 2700 is essentially set up in a manner similar to that described above
19 with respect to the video projects. That is, for each track, a row is added to the
20 grid and a grid entry is made for the time period during which the source on that
21 track desires to be routed to the primary output of the matrix switch. In the
22 present example, grid entries are made for sources A-E. Next, in the same way
23 that a transition or effect was allocated a row in the grid, a "mix" element is
24 allocated a row in the grid as shown and a grid entry is made to indicate that the
25 mix element desires to be routed to the primary output of the matrix switch for a

1 period of time during which two or more sources compete for the matrix switch's
2 primary output. Note that in this embodiment, allocation of a grid row for the mix
3 element can be implied. Specifically, whereas in the case of a video project,
4 overlapping sources simply result in playing the higher priority source (unless the
5 user defines a transition between them), in the audio realm, overlapping sources
6 are treated as an implicit request to mix them. Thus, the mix element is allocated a
7 grid row any time there are two or more overlapping sources.

8 Once the mix element is allocated into the grid, the grid is processed to
9 redirect any conflicting source entries to matrix switch output pins that correspond
10 to the mix element. In the above case, redirection of the grid entries starts with pin
11 3 and proceeds through to pin 7. The corresponding matrix switch is shown in
12 Fig. 28. Notice that all of the sources are now redirected through the mix element
13 which is a multi-input/one output element. The mix element's output is fed back
14 around and becomes input pin 15 of the matrix switch. All of the programming of
15 the matrix switch is now reflected in the grid 2700. Specifically, for the indicated
16 time period in the grid, each of the sources is routed to the mix element which, in
17 turn, mixes the appropriate audio streams and presents them to the primary output
18 pin 0 of the matrix switch.

19 20 Compositions

21 There are situations that can arise when building an editing project where it
22 would be desirable to apply an effect or a transition on just a subset of a particular
23 project or track. Yet, there is no practicable way to incorporate the desired effect
24 or transition. In the past, attempts to provide added flexibility for editing projects
25 have been made in the form of so called "bounce tracks", as will be appreciated

1 and understood by those of skill in the art. The use of bounce tracks essentially
2 involves processing various video layers (i.e. tracks), writing or moving the
3 processed layers or tracks to another location, and retrieving the processed layers
4 when later needed for additional processing with other layers or tracks. This type
5 of processing can be slow and inefficient.

6 To provide added flexibility and efficiency for multi-media editing projects,
7 the notion of a *composite or composition* is introduced. A composite or
8 composition can be considered as a representation of an editing project as a single
9 track. Recall that editing projects can have one or more tracks, and each track can
10 be associated with one or more sources that can have effects applied on them or
11 transitions between them. In addition, compositions can be nested inside one
12 another.

13 14 Example Project with Composite

15 Consider, for example, Fig. 29 which illustrates an exemplary project 2900
16 having a composition 2902. In this example, composition 2902 comprises sources
17 B and C and a transition between B and C that occurs between $t = 12-14$. This
18 composition is treated as an individual track or layer. Project 2900 also includes a
19 source A, and a transition between source A and composition 2902 at $t = 4-8$. It
20 will be appreciated that compositions can be much more complicated than the
21 illustrated composition, which is provided for exemplary purposes only.
22 Compositions are useful because they allow the grouping of a particular set of
23 operations on one or more tracks. The operation set is performed on the grouping,
24 and does not affect tracks that are not within the grouping. To draw an analogy, a
25 composition is similar in principle to a mathematical parenthesis. Those

1 operations that appear within the parenthesis are carried out in conjunction with
2 those operations that are intended to operate of the subject matter of the
3 parenthesis. The operations within the parenthesis do not affect tracks that do not
4 appear within the parenthesis.

5 In accordance with the processing that is described above in connection
6 with Fig. 19, a first data structure is defined that represents the editing project.
7 Fig. 30 shows an exemplary data structure 3000 in the form of a hierarchical tree
8 structure. In this example, group node 3002 includes two children—track node
9 3004 and composite node 3006. Track node 3004 is associated with source A.
10 Composite node 3006 includes two children—track nodes 3008 and 3010 that are
11 respectively associated with sources B (3008a) and C (3010a). A transition T2
12 (3012) is applied on source C and a transition T1 (3014) is applied on composition
13 3006.

14 Next, data structure 3000 is processed to provide a second data structure
15 that is configured to program the matrix switch. Note that as the data structure is
16 being programmed, a matrix switch is being built and configured at the same time.
17 In this example, the second data structure comprises a grid structure that is
18 assembled in much the same way as was described above. There are, however,
19 some differences and, for purposes of understanding, the complete evolution of the
20 grid structure is described here. In the discussion that follows, the completed
21 matrix switch is shown in Fig. 38.

22 When the rendering engine initiates the depth-first, left-to-right traversal of
23 data structure 3000, the first node it encounters is track node 3004 which is
24 associated with source A. Thus, a first row of the grid is defined and a grid entry
25

1 is made that represents the time period for which source A desires to be routed to
2 the matrix switch's primary output pin.

3 Fig. 31 shows the state of a grid 3100 after this first processing step. Next
4 the traversal of data structure 3000 encounters the composite node 3006. The
5 composite node is associated with two tracks—track 3008 and track 3010. Track
6 3008 is associated with source B. Accordingly, a second row of the grid is defined
7 and a grid entry is made that represents the time period for which source B desires
8 to be routed to the matrix switch's primary output pin. Additionally, since B is a
9 member of a composition, meta-information is contained in the grid that indicates
10 that this grid row defines one boundary of the composition. This meta-
11 information is graphically depicted with a bracket that appears to the left of the
12 grid row.

13 Fig. 32 shows the state of grid 3100 after this processing step. Next, the
14 traversal of data structure 3000 encounters node 3010 which is associated with
15 source C. Thus, a third row of the grid is added and a grid entry is made that
16 represents the time period for which source C desires to be routed to the matrix
17 switch's primary output pin.

18 Fig. 33 shows the state of grid 3100 after this processing step. Notice that
19 the bracket designating the composition now encompasses the grid row associated
20 with source C. The traversal next encounters node 3012 which is the node
21 associated with the *second* transition T2. Thus, as in the above example, a grid
22 row is added for the transition and a grid entry is made that represents the time
23 period for which the transition desires to be routed to the matrix switch's primary
24 output pin.

1 Fig. 34 shows the state of grid 3100 after this processing step. Notice that
2 the bracket designating the composition is now completed and encompasses grid
3 row entries that correspond to sources B and C and the transition between them.
4 Recall from the examples above that a transition, in this example, is programmed
5 to operate on two inputs and provide a single output. In this instance, and because
6 the transition occurs within a composition, the transition is constrained by a rule
7 that does not allow it to operate on any elements outside of the composition.
8 Thus, starting at the transition entry and working backward through the grid,
9 entries at the same tree level and within the composition (as designated by the
10 bracket) are examined to ascertain whether they contain entries that indicate that
11 they want to be routed to the output during the same time that the transition is to
12 be routed to the output. Here, both of the entries for sources B and C have
13 portions that conflict with the transition's entry. Accordingly, those portions of
14 the grid entries for sources B and C are redirected or changed to correspond to
15 output pins that are associated with a transition element that corresponds to
16 transition T2.

17 Fig. 35 shows the state of grid 3100 after this processing step. The
18 traversal next encounters node 3014 which is the node that is associated with the
19 transition that occurs between source A and composition 2902 (Fig. 29).
20 Processing of this transition is similar to processing of the transition immediately
21 above except for the fact that the transition does not occur within the composition.
22 Because the transition occurs between the composition and another source, one of
23 the inputs for the transition will be the composition, and one of the inputs will be
24 source A (which is outside of the composition). Thus, a grid row is added for this
25

1 transition and a grid entry is made that represents the time period for which the
2 transition desires to be routed to the matrix switch's primary output pin.

3 Fig. 36 shows the state of grid 3100 after this processing step. At this point
4 then, the grid is examined for entries that conflict with the entry for transition T1.
5 One conflicting grid entry is found for the row that corresponds to source B (inside
6 the composition) and one that corresponds to source A (outside the composition).
7 Accordingly, those portions of the grid row that conflict with transition T1 are
8 changed or redirected to have values that are associated with output pins of the
9 matrix switch that are themselves associated with a transition element T1. In this
10 example, redirection causes an entry of "3" and "4" to be inserted as shown.

11 Fig. 37 shows the state of grid 3100 after this processing step. If necessary,
12 a pruning operation would further ensure that the grid has no competing entries for
13 the primary output of the matrix switch. The associated input pin numbers of the
14 matrix switch are shown to the left of grid 3100.

15 Fig. 38 shows a suitably configured matrix switch that has been build in
16 accordance with the processing described above. Recall that, as data structure
17 3000 (Fig. 30) is processed by the rendering engine, a matrix switch is built and
18 configured in parallel with the building and processing of the grid structure that is
19 utilized to program the matrix switch. From the matrix switch and grid 3100 of
20 Fig. 37, the programming of the switch can be easily ascertained.

21 Fig. 38a shows an exemplary data structure that represents a project that
22 illustrates the usefulness of composites. In this example, the project can
23 mathematically be represented as follows:

24
25 (Fx-noisy (A Tx-Blend B)) Tx-Blend C

1
2 Here, an effect (noisy) is applied to A blended with B, the result of which is
3 applied to a blend with C. The composite in this example allows the grouping of
4 the things beneath it so that the effect (noisy), when it is applied, is applied to
5 everything that is beneath it. Notice that without the composite node, there is no
6 node where an effect can be applied that will affect (A Tx-Blend B). Hence, in
7 this example, operations that appear within the parenthesis are carried out on
8 tracks that appear within the parenthesis. Those operations do not affect tracks
9 that are not within the parenthesis.

10 Fig. 39 is a flow diagram that described steps in a method in accordance
11 with one embodiment. The method can be implemented in any suitable hardware,
12 software, firmware, or combination thereof. In the presently-described example,
13 the method is implemented in software.

14 Step 3900 defines a multimedia editing project that includes at least one
15 composite. The composite represents multiple tracks as a single track for purposes
16 of the processing described just below. It is important to note that, in the
17 processing described just below, and because of the use of composites, the extra
18 processing that is required by bounce tracks is avoided (i.e. operating on two
19 tracks, moving the operation result to another location, and retrieving the
20 operation result when later needed). This reduces the processing time that is
21 required to render a multi-media project. Step 3902 defines a first data structure
22 that represents the editing project. Any suitable data structure can be utilized. In
23 the present example, a data structure in the form of a hierarchical tree is utilized.
24 An exemplary tree is shown in Fig. 30. Step 3904 processes the first data structure
25 to provide a second data structure that is configured to program a matrix switch.

1 In the illustrated example, the second data structure comprises a grid structure.
2 Exemplary processing is described in the context of Figs. 30-37. Step 3906 then
3 programs the matrix switch using the second data structure.
4

5 **Source Combining**

6 Having introduced the various architectural and implementation elements
7 of the present invention, above, attention is now drawn to Figs. 40-43, wherein
8 another aspect of the illustrated embodiment is presented. As introduced above,
9 the opening and processing of media represents consumption of memory and
10 processing resources. Thus, performance improvements may be achieved by
11 reducing the number of times a source is accessed. Thus, a method is presented in
12 accordance with one aspect of the present invention that serves to reduce the
13 number of times a source is accessed, e.g., a method of source combining. It is to
14 be appreciated, however, that the following is but one example implementation of
15 the broader inventive concept of reducing the number of times a source need be
16 accessed during execution of a development project. Alternative methods of
17 source combining of greater or lesser complexity may well be used within the
18 spirit and scope of the present invention. Indeed, such alternative methods are
19 anticipated within the scope of the present invention.

20 **Fig. 41** illustrates an example method of generating a filter graph, in
21 accordance with one aspect of the present invention. As shown, method 4000
22 begins with block 4002, wherein render engine 222 receives an indication to
23 generate a development project. According to one implementation, as discussed
24 above, render engine 222 receives the indication from a higher-level application
25

1 216, e.g., media processing application 224, to assist a user in generating a
2 processing project (e.g., a media processing project).

3 In block 4004, render engine 222 identifies the number and nature of the
4 media sources within the user-defined processing project, in preparation for
5 generating a filter graph representation of the processing project. As introduced
6 above, for each of the identified sources, render engine 222 determines the
7 necessary transform filters 306 required to pre-process the source (i.e., source
8 chain), preparing the processing chain for presentation to the matrix switch filter
9 308 and one or more transition/effect filters 306. Unlike conventional
10 implementations, which would proceed to generate the entire filter graph in
11 preparation for execution of the processing project, render engine 222 generates a
12 list of sources and when they are required in the filter graph. According to one
13 implementation, the list is referred to as a reuse list, and is maintained within
14 render engine 222. An example of a data structure comprising a reuse list is
15 presented with reference to Fig. 41.

16 Turning briefly to Fig. 41, a graphical illustration of an example data
17 structure comprising a source reuse list is presented. As shown, the reuse list 4100
18 is comprised of a number of information fields, e.g., 4102-4110 which detail, in
19 part, the relationship between clips in a track. More particularly, the reuse list
20 4100 is shown comprising a track identification field 4102, a source identification
21 field 4104, a project time field 4106 and a source time field 4108.

22 Upon identifying a project source and the associated filters required for pre-
23 processing the source (i.e., the source chain), render engine 222 assigns each track
24 an identifier which uniquely identifies the source track within the context of the
25 filter graph. In this regard, reuse list 4100 includes a field 4102 which maintains a

1 list of tracks utilized in the associated project. In accordance with the illustrated
2 example paradigm of the media processing system, the track identifier is utilized
3 to represent a media clip from a given source.

4 The source identifier field 4104 contains information denoting the project
5 source associated with a particular track identifier. In this regard, the source
6 identifier field 4104 may well contain a file name, a file handle, or any other
7 suitable source identifier.

8 The project time field 4106 denotes at what point during project execution
9 the media clip is required. The source time field 4108 denotes what portion of the
10 source file is required to support execution of the processing project. It should be
11 appreciated that a user may well utilize the whole source file or any part thereof,
12 as defined by the processing project.

13 In accordance with the illustrated example implementation of Fig. 41, two
14 tracks are depicted 4110 and 4112. As shown, each of the tracks represent media
15 from a common source (e.g., source ID 4213) and, the source media clips are
16 adjacent to one another in the project (e.g., project time 4106) as well as within the
17 source file (e.g., source time 4108). As will be developed more fully below,
18 source clips may well be combined in certain situations into a single clip, as is
19 represented by track 4114 in Fig. 41. It is to be appreciated that, although depicted
20 as a two-dimensional data structure, reuse lists of greater or lesser complexity may
21 well be substituted without deviating from the spirit and scope of the present
22 invention.

23 Returning to Fig 40 and, in particular, block 4006, render engine 222
24 reduces the number of source accesses where possible, in accordance with one
25 aspect of the present invention. More particularly, render engine 222 analyzes the

1 reuse list 4100 to identify opportunities to reduce the number of source accesses
2 by combining source clips which meet certain criteria. According to one
3 implementation, the criteria used by render engine 222 include one or more of (1)
4 the source clips must occur next to one another in the project, (2) the clips appear
5 next to one another in the source, and (3) the clips must share a common pre-
6 processing source chain (i.e., must require the same pre-processing). If this
7 criteria is met, render engine 222 may combine the clips into a single clip. More
8 specifically, render engine 222 modifies the reuse list 4100 (Fig. 41) to replace the
9 multiple source accesses (4110, 4112) with a single source access 4114
10 representing both source accesses as a single access. It is to be appreciated that
11 removing a source access improves filter graph performance and, accordingly, the
12 perceived performance of the development system by the user.

13 In block 4008, once render engine 222 has reduced the number of source
14 file accesses (block 4006), render engine 222 dynamically generates and manages
15 the filter graph to support execution of the development project. In accordance
16 with one aspect of the present invention, render engine 222 invokes only those
17 source chains associated with sources that are necessary to support the current
18 and/or impending execution of the filter graph. It is to be appreciated that by not
19 opening each of the chains of a processing project, render engine 222 reduces the
20 amount of memory required to build the filter graph, thereby reducing the amount
21 of memory required to complete execution of the project.

22 Turning to **Fig. 42**, an example method for source combining is presented,
23 in accordance with one aspect of the present invention. As shown, the method
24 begins with block 4202, wherein render engine 222 identifies adjacent clips from a
25 common source, i.e., project aligned clips. As introduced above, render engine

1 222 analyzes the reuse list 4100 to identify all of the clips associated with a
2 particular track. For each of the adjacent source clips within a track, render engine
3 222 determines whether the clips are adjacent to one another with respect to their
4 source time, block 4204. That is, identifying that the source clips are adjacent to
5 one another by project time (e.g., occurring within the same track), render engine
6 222 next determines whether the clips are adjacent to one another in the source
7 file, i.e., whether the clips are source aligned. According to one implementation, if
8 the source clips are project and source time aligned, render engine 222 determines
9 whether the source clips share a common preprocessing source chain (i.e., the
10 require the same pre-processing).

11 If the source clips are source aligned, render engine 222 next determines
12 whether the clips require unique pre-processing (e.g., decoding, frame rate
13 conversion, sizing, etc.), block 4206. If unique pre-processing is required (block
14 4206), or the adjacent project clips are not source aligned (block 4204), the source
15 clips will require an independent source chain of filters and, thus cannot be source
16 combined. Accordingly, the source clips are accessed and processed
17 independently, block 4208.

18 If, in block 4206, render engine 222 determines that the clips are source
19 aligned and each share common pre-processing requirements, render engine 222
20 combines the adjacent source clips into a single clip and updates the reuse list
21 4100 accordingly, block 4210 (e.g., clip 4114 of Fig. 41). The combined source
22 clip is representative of each of the otherwise individual clips, while requiring a
23 single source processing chain and, thus, a single source access. In block 4212,
24 render engine 222 determines whether all of the clips of the reuse list has been
25

1 analyzed and, if not, the process continues in an iterative manner with block 4202
2 until the entire reuse list has been analyzed and appropriate source clips combined.

3 **Fig. 43** graphically illustrates the source combining aspect of the present
4 invention. As shown, Fig. 43 illustrates a project 4300 of two tracks of clips (e.g.,
5 clips 4302-4318, and 4320-4324) separated by a transition 4318. In accordance
6 with the source combining aspect of the present invention, introduced above,
7 render engine 222 analyzes a reuse list 4100 representation of project 4300 to
8 identify source clips which may be combined. As introduced above, render engine
9 222 will combine source clips which are project and source time aligned, and
10 which do not require unique pre-processing.

11 In the illustrated example of Fig. 43, clips 4302 and 4304 are project
12 aligned and source aligned. If such clips do not require independent pre-
13 processing, they are combined into a single clip 4326 by render engine 222. Note
14 that although clips 4304 and 4308 are project aligned, they are not source aligned
15 (e.g., the media end time (9) of clip 4304 does not abut the media start time (10) of
16 clip 4308), i.e., there is a gap of one elemental unit (e.g., a second of time). The
17 process of source combining is performed for other clips in the development
18 project, reducing the total number of source clips from eleven to six in the
19 illustrated example of Fig. 43. Thus, it is to be appreciated that the source
20 combining aspect of the present invention represents another feature which
21 reduces the computational and memory requirements necessary to support even
22 the most complex development projects.

23
24
25

Source Filter Reuse

As introduced above, conventional media processing systems may generate an individual thread each time content was required from a source, even if the source had been accessed earlier. This redundant loading/unloading of a source is computationally expensive, and consumes precious memory resources. Extending the concept of source combining introduced above, a filter and related methods for sharing a common source and source filter among multiple processing threads will now be introduced, in accordance with yet another aspect of the present invention.

Turning to **Fig. 44**, a block diagram of an example filter graph 4400 is presented incorporating a segment filter 4406 which, as will be shown, dynamically couples a source filter to one or more processing chain in accordance with the teachings of the present invention. In accordance with the illustrated example of Fig. 44, filter graph 4400 is depicted comprising a source 4402, one or more pre-processing transform filters 4404, a segment filter 4406 and one or more pre-processing transform filter(s) 4408A-N, each coupled to a matrix switch 308 and rendering filter(s) 4410, 4412, respectively.

As used herein, segment filter 4406 is designed to sit between a source filter and matrix switch 308 to provide multiple processing chains with source content from a single source, where it is impossible to combine the source clips (as introduced above). Render engine 222 invokes an instance of the segment filter 4406 after the greatest common pre-processing filter 4404 for each of the chains. That is, each of the processing chains may require the source content in a different format (e.g., size, frame rate, decode format, etc.). To the extent that the chains share common pre-processing attributes, those filter(s) (4404) are placed before the segment filter 4406 where practicable. In many instances, none of the chains

1 share common pre-processing and the pre-processing filter(s) merely comprise the
2 source filter.

3 The segment filter 4406 acts as a controller, or throttle for the source,
4 instructing the source filter to deliver content from source 4402 at select times.
5 According to one implementation, the segment filter 4406 is, in turn, controlled by
6 the render engine 222 and/or the matrix switch filter 308 to provide select content
7 at select times on select inputs of the matrix switch filter 308. According to one
8 implementation, the segment filter 4406 issues a “seek” command to the source
9 filter to request particular content from the source. The source filter then delivers
10 the requested content through the segment filter 4406 and appropriate pre-
11 processing filter(s) 4408A-N to deliver the desired content to the requesting matrix
12 switch 308 to support processing of the development project.

13 As introduced above, render engine 222 is responsive to higher-level user
14 interfaces, e.g., applications 224. In this regard, it is possible that the filter graph
15 will receive user-commands while the filter graph is executing the development
16 project. In accordance with the media processing system paradigm, for example,
17 it is foreseeable that a user-invoked seek will be received by the filter graph during
18 execution of the development project. Such user defined commands are typically
19 serialized with commands issued by filters within the filter graph during the
20 normal course of execution. In accordance with the illustrated example
21 implementation, where matrix switch 308 “throttles” execution of the filter graph,
22 matrix switch 308 issues a seek command of its own to the source filter, requesting
23 the information desired by the user. According to an alternate embodiment, seeks
24 received from a higher-level application (and, therefore, representative of a user
25 command) are afforded a higher priority within the filter graph. In such an

1 implementation, all segment filters 4406 residing within the filter graph are also
2 notified of such high-priority seeks, so that they can identify what content they
3 will be required to provide next and, therefore, issue a revised seek command of
4 their own.

5 The remaining pre-processing transform filter(s) 4408A-N, matrix switch
6 filter(s) 308 and rendering filter(s) 4410 each function as described above.

7 Turning now to Fig. 45, an example method for generating a filter graph is
8 presented incorporating the teachings of the present invention. More particularly,
9 the method of Fig. 45 is similar to the method of Fig. 42 wherein render engine
10 222 attempted source combining of source clips, which were not project and
11 source time aligned, or which required unique pre-processing of some sort. In Fig.
12 42, however, if the source clips were not source time aligned (4204) and/or the
13 clips required separate pre-processing (block 4206), each clip was assigned to a
14 separate processing chain. In Fig. 45, however, this problem is resolved with
15 introduction of a segment filter 4406.

16 More specifically, with reference to Fig. 45, render engine 222 identifies
17 multiple source clips from a common source which are not source time aligned,
18 block 4204 and/or require separate pre-processing filter(s), block 4206. Render
19 engine 222 generates a segment filter 4406 for the filter graph to reuse the source
20 and at least the source filter, block 4502. That is, the render engine 222 inserts a
21 segment filter 4406 between the source filter and one or more processing chains to
22 selectively provide otherwise disparate source clips from a single source. But for
23 use of the segment filter 4406 in the filter graph, the method 4500 of Fig. 45
24 executes in a fashion similar to Fig. 42, above.
25

Turning now to Fig. 46, a flow chart of an example method of segment filter operation is presented, in accordance with one example implementation of this aspect of the present invention. In accordance with the illustrated example embodiment of Fig. 46, the method begins in block 4602, wherein the segment filter 4406 seeks the source to the place that source data is first needed. As introduced above, segment filter 4406 receives a request for source content from matrix switch filter 308. It should be appreciated that insofar as segment filter 4406 may well support a plurality of processing chains coupled to a plurality of matrix switch filters 308, a number of such requests may be received simultaneously. According to one implementation, each of the matrix switch filters 308 assigns a priority to the request for source content, wherein the priority of the request changes as the time the content is needed draws near. According to an alternate implementation, render engine 222 determines *a priori* whether source content will be required simultaneously and, if so, provides a separate source chain to accommodate such simultaneous content requests, thereby eliminating the situation of the segment filter 4406 receiving simultaneous requests.

In block 4604, the source filter retrieves the requested content and passes the data to the switch until some sort of indication is received that the end of content has been received (e.g., an end-of-stream (EOS) indication, an application interrupt, etc.). As introduced above, an application interrupt may be issued when a user, through a user interface (e.g., media control application 224), wants to seek to a certain point in the development project.

In block 4606, segment filter 4406 determines whether an EOS or an application interrupt is received. If not, the process continues with block 4604. If so, segment filter 4406 identifies the next required segment and when it will be

1 required, given the current seek location received from the matrix switch filter
2 308. Based, at least in part, on the current seek location, segment filter 4406
3 determines whether more segments of the source are required, block 4610. As
4 introduced above, if a user-defined seek command is issued, it may be issued to a
5 location in the development project where no further content is required from a
6 particular source. Thus, segment filter 4406 determines whether additional
7 segments are required in block 4610.

8 If no further segments are required along one of the processing chains
9 leading from the segment filter, render engine 222 may remove (at least
10 temporarily) that chain from the filter graph to free memory space and a matrix
11 switch filter input for other processing chains, block 4612.

12 If, in block 4610 further segments are required, segment filter 4406 issues a
13 seek instruction directing the source filter to retrieve and deliver the next segment,
14 in accordance with the matrix switch filter instructions, block 4614. This process
15 continues in an iterative fashion with block 4604.

16 Although the invention has been described in language specific to structural
17 features and/or methodological steps, it is to be understood that the invention
18 defined in the appended claims is not necessarily limited to the specific features or
19 steps described. Rather, the specific features and steps are disclosed as preferred
20 forms of implementing the claimed invention.
21
22
23
24
25